Production Planning, Monitoring & Control

A Domain Model, a Set of Requirements and a Set of High-level Software Designs

Dines Bjørner
Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Richard Petersens Plads, Building 322
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark

db@imm.dtu.dk, http://www.imm.dtu.dk/~db

January 12, 2006

Preface

We first present a narrative and a formalisation of core issues of manufacturing; more specifically we attempt to answer such questions as *what are products, what is production, what is production planning, what is production monitoring and control, etc.?*

More generally we develop, in this report and according to the triptych principles of software engineering [1, 2, 3] a variety of software systems for manufacturing. Chapter 2 unfolds a domain model for manufacturing; Chap. 3 unfolds a variety of requirements models, and Chap. 4 develops this variety of requirements prescriptions into respective software packages.

A very specific "novelty" of this report is the exploration of the close interface between computing science and operations research: optimisation of allocation and scheduling of production resources. This aspect of the present report is the work primarily of Prof. Jens Clausen and an MSc student of ours, José María Carabús Rodríguez during the latters MSc thesis work, spring and summer 2006.

Contents

Chapter 1

Triptych Development of Manufacturing Software

1.1 Manufacturing

By *manufacturing* we mean (i) the construction of artifacts [i.e., man-made (or machine-made) products, or part products (i.e., parts)], (ii) from (other) parts (other artifacts), (iii) according to specifications, (iv) by means of machines (which [possibly helped by human workers] mechanically follow the specifications), and (iv) in response to customer orders.

Thus there are a number of phenomena involved in *manufacturing*. They are *parts, specifications, machines* and *orders*.

Parts are

Specifications are

Machines are

Orders are

1.2 The TripTych Paradigm of Software Development

There is the dogma:

- Before software can be designed its requirements must be understood.
- Before requirements can be prescribed the underlying application domain must be understood.
- To understand the requirements their prescription must both
  - ⋆ be in the natural, professional language of the domain and of software engineering
  - ⋆ and must be formalised.

- To understand the domain its description must both
  - ⋆ be in the natural, professional language of the domain
  - ⋆ and must be formalised.

And there are therefore the consequences of the dogma:

- In software development
  - ⋆ we must first — and carefully — describe the domain,
  - ⋆ then, based on the domain description, prescribe the requirements,
  - ⋆ and, finally, based on the requirements prescription, design the software.
- The phases outlined above are linked:
  - ⋆ precise, formal relations can be established between the domain description and the requirements prescription, and
  - ⋆ precise, formal relations can be established between the requirements prescription and the software specification.

The consequences, we believe, of adhering to the above phase-wise development. are:

- That the emphasis
  - ⋆ on domain prescription as a prerequisite for requirements prescription secures the right computing system;
- and the emphasis
  - ⋆ on requirements prescription as a prerequisite for software specification secures that the computing system is right.

1.3 Aims of This Report

The aims of this report are twofold:

- to illustrate, to the reader, the essentials of the triptych paradigm of software engineering including what we consider the benefits of formalisation, and
- to illustrate, independently of the larger scope of the full development, the more narrow, but also, by far, the more novel aspect of the span of domain description.

That is, the really new thing here is the wide-ranging description which soon follows, of a description, informal and formal, of a domain, in this case the multi-modal transport domain.

1.4 Objectives of This Report

The objectives of this report are correspondingly twofold:

- to hopefully convince the reader that that is how software ought be professionally developed: first domain description, then requirements prescription and finally software specification (design and coding), and
- to hopefully further spur an area of research for computing science, here called domain theories or domain engineering.

1.5 Structure of This Report

It should therefore come as no surprise that the structure of this report follows the dogma outlined above. The next three chapters shall therefore outline a domain description of manufacturing, a small variety of requirements, and a similar variety of software designs.

The main reference to the triptych approach is [1, 2, 3].

Chapter 2

A Domain Model of Manufacturing

2.1 Parts

1. An atomic part is a smallest unit of man-made production.

   type
   P
   value
   is_atomic_P: P → **Bool**

2. An atomic part has a part number, as has all parts, whether atomic or composite.

   type
   Pn
   value
   obs_Pn: P → Pn

3. A composite part consists of two or more parts which have been manufactured (fitted, assembled, welded, etc.) together according to some mereology.

   value
   is_composite_P: P → **Bool**
   axiom
   ∀ p:P•is_atomic_P(p)∧∼is_composite_P(p) ∨ is_composite_P(p)∧∼is_atomic_P(p)

4. A composite part may have two or more occurrences (components) of parts of the same part number. Mereologically they appear (occur) in distinct "locations" of the whole part. We abstract locations (etc.) by associating with each part a unique identification, $\pi : \Pi$.

type
  Π
  COMPS = P-**set**
value
  obs_Π: P $\to$ Π
  obs_COMPS: P $\to$ COMPS

  no_of_occurrences: P $\times$ P $\xrightarrow{\sim}$ **Nat**
  no_of_occurrences(cp)(p) $\equiv$
    card $\{p'|p':P \bullet p' \in$ obs_COMPS(cp) $\wedge$ obs_Pn(p')=obs_Pn(p)$\}$
    pre: is_composite_P(cp)$\wedge$p $\in$ obs_COMPS(cp)
axiom
  $\forall$ p:P $\bullet$
    is_atomic_P(p) $\Rightarrow$ obs_COMPS(p)=$\{\}$ $\wedge$
    is_composite_P(p) $\Rightarrow$
      obs_COMPS(p)$\neq\{\}$ $\wedge$
      $\forall$ p$'$,p$''$:P $\bullet$ $\{p',p''\}\subseteq$**dom** obs_COMPS(p) $\wedge$ p$'\neq$p$''$ $\Rightarrow$
        obs_Π(p$'$) $\neq$ obs_Π(p$''$)

5. From the above we observe that no two physically manifest parts are identical: They may be of the same kind, i.e., part number, but they will, as a "law of nature" have distinct identifications. We are not saying that these identifications are physically manifest things, i.e., "labels" the part. We are saying that these identifications are comcepts.[1]

2.2 Machines

6. A machine basically offers a function which takes a non-zero number of parts and produces a non-zero number of other, distinct parts.

type
  Parts$'$ = P-**set**
  Parts = $\{|$ parts:Parts$'$ $\bullet$ parts$\neq\{\}$ $|\}$
  MOp
  MFct = Parts $\to$ Parts
value
  obs_MFct: MOp $\to$ MFct
axiom
  $\forall$ mop:MOp $\bullet$ $\{$obs_Pn(p)$|$p:P$\bullet$p $\in \mathcal{D}\}$ $\cap$ $\{$obs_Pn(p)$|$p:P$\bullet$p $\in \mathcal{D}\}$ = $\{\}$

[1] It is like the coins in your pocket: There may be several instances of a shilling but they are all distinct in space (and otherwise).

$\mathcal{D}$, the definition set function, is a non-computable function. So is $\mathcal{R}$, the range set function.

7. We can characterise the functionality of a machine by the signature of the machine function mop:MOp.

type
    Config′ = Pn $\overrightarrow{m}$ **Nat**
    Config = {| c:Config′ • 0∉ **rng** c |}
    Input,Output = Config
    MSig′ = Input × Output
    MSig = {| (i,o):MSig′ • **dom** i ∩ **dom** o = {} |}

8. A machine proper can pragmatically be thought of as an "optional formal machine and zero, one or more workers". If the "optional formal" machine is not there, then the machine "embodies" at least one worker. In any case we consider the "optional formal machine and zero, one or more workers" as a unit. The machine, in addition to its machine operation[2], can be thought of as also being represented by the machine signature and a machine in-tray and a machine out-tray. The in- and out-trays, at one one moment, consists of zero, one or more parts. The parts may, or may not be relevant to the machine operation. Usually they are. We shall, in fact, expect that the in-tray [out-tray] parts are of the kind (i.e., having the input [output] part numbers) of the machine signature.

type
    MC
    MACH′ = Input × MSig × MOp × Output
    MACH = {| mach:MACH′ • wf_MACH(mach) |}
value
    obs_MACH: MC → MACH
    wf_MACH(input,(isig,osig),op,output) ≡
        dom input ⊆ **dom** isig ∧
        dom input = $\mathcal{D}$(op) ∧
        dom output = $\mathcal{R}$(op) ∧
        dom output ⊆ **dom** osig

9. Figure 2.1 on the following page intends to illustrate that a machine can be considered to consist of an in tray, a facility for performing the machine operation and an out tray.

[2]The machine operation is performed either by the "optional formal machine" or the "optional formal machine and one or more workers" or, when the "optional formal machine" is not there, the "one or more workers".

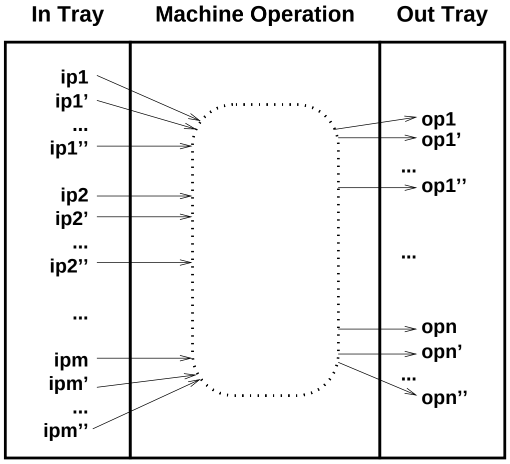

Figure 2.1: A schematic machine

2.3 Machine Operation

10. Figure 2.1 also intends to show that a machine operation consumes one or more occurrences (ip1, ip1′, ..., ip1″) of parts of one part number (ip_1), one or more occurrences (ip2, ip2′, ..., ip2″) of parts of another part number (ip_2), etc., and one or more occurrences (ipm, ipm′, ..., ipm″) of parts of yet another part number (ip_m),

11. For a machine operation to take place it must be enabled. A machine is enabled if the in-tray contains at least the number of parts for each of the parts required in the machine operation, that is, as designated in the machine operation signature.

$$
\begin{array}{l}
\textbf{value} \\
\quad \text{is\_enabled: MACH} \rightarrow \textbf{Bool} \\
\quad \text{is\_enabled(input,(isig,osig),op,output)} \equiv \\
\qquad \textbf{dom}\ \text{input} = \textbf{dom}\ \text{isig} \wedge \\
\qquad \forall\ \text{pn:Pn} \bullet \text{pn} \in \textbf{dom}\ \text{input} \Rightarrow \text{input(pn)} \geq \text{isig(pn)}
\end{array}
$$

12. An enable machine can fire. Firing means that the machine performs its function: consumes an appropriate number of parts (removes them) from the in-tray and produces another appropriate number of parts (adds them) to the out-tray.

$$
\begin{array}{l}
\textbf{value} \\
\quad \text{fire: MACH} \rightarrow \text{MACH} \\
\quad \text{fire(input,(isig,osig),op,output)}\ \textbf{as}\ \text{(input}',\text{(isig}',\text{osig}'\text{),op}',\text{output}'\text{)} \\
\qquad \textbf{pre}\text{: is\_enabled(input,(isig,osig),op,output)} \\
\qquad \textbf{post}\text{: isig}' = \text{isig} \wedge \text{osig}' = \text{osig} \wedge \text{op}' = \text{op} \wedge \\
\qquad\qquad \text{input}' = \text{input} \setminus \mathcal{D}\,\text{op} \wedge \text{output}' = \text{output} \cup \mathcal{R}\,\text{op}
\end{array}
$$

2.4 Production Floors

13. A production floor of a manufacturing plant consists of a non-zero number of uniquely identified machines.

    type
    PF = Mid $\overrightarrow{m}$ MC

14. We can think of a production floor as shown in Fig. 2.2.

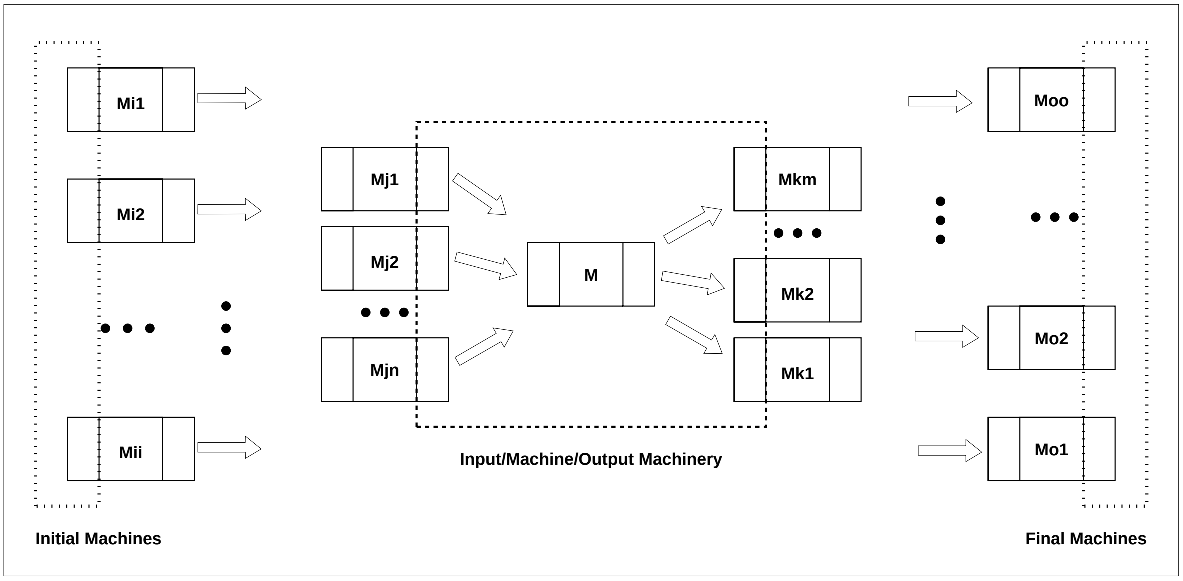

Figure 2.2: A schematic production floor

15. Let us focus on the entral — what we call — the Input/Machine/Output Machinery in that figure. Machine M potentially receives one or more parts of possibly different part numbers from machine M_{j_1}, one or more parts of possibly different part numbers from machine M_{j_2}, etc., and one or more parts of possibly different part numbers from machine M_{j_n}. And machine M potentially delivers one or more parts of possibly different part numbers to machine M_{k_1}, one or more parts of possibly different part numbers to machine M_{k_2}, etc., and one or more parts of possibly different part numbers to machine M_{k_m}. We say "potentially" since, as wee shall see later, machine M may receive or deliver parts from, respectively to other machines.

    For a machine to "potentially receive" parts from another machine means two things: the parts received are necessary for machine M to perform its operation, i.e., are required inputs to M.

    For a machine to "potentially deliver" parts to another machine means two things: the parts delivered are produced by machine M, i.e., are output from M.

16. Figure 2.3 shows a more general situation for the input/machine/output machinery of Fig. 2.2 on the previous page.

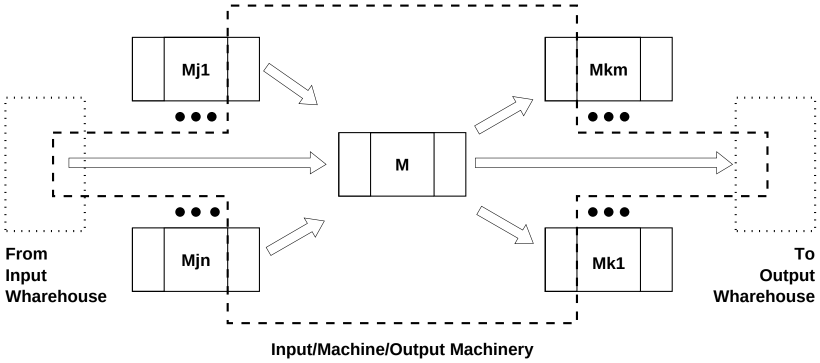

Figure 2.3: A general input/machine/output machinery

Whereas Fig. 2.2 on the preceding page could be construed as expressing that all inputs to the central machine M came from other machines, Fig. 2.3 hints at the possibility that some, or all, parts input to a machine operation may come from an input wharehouse.

The same wrt. outputs. Instead of all central machine M outputs being delivered to other machines, some, or all, may go to an output wharehouse.

17. It is thus that we arrive at a production unit consisten of a production floor and an input and an output wharehouse.

type
  InWh,OutWh = P-**set**
  PU = InWh × PF × OutWh

18.

19.

20.

21.

22.

23.

24.

25.

26.

27.

28.

29.

30.

31.

32.

2.5

33.

34.

35.

36.

37.

38.

39.

40.

41.

42.

2.6

43.

44.

45.

46.

47.

48.

49.

50.

51.

52.

2.7

53.

54.

55.

56.

57.

58.

59.

60.

61.